Propositional Satisfiability (SAT) and SAT-based decision procedures

Marco Maratea
j.w.w. Enrico Giunchiglia

Laboratory of Systems and Technologies for Automated Reasoning (STAR-Lab)
Dipartimento di Informatica, Sistemistica e Telematica (DIST)

L'Aquila, 09 Apr 2008

Motivation

1. Propositional satisfiability (SAT) is one of the most studied fields in AI and CS
2. Very efficient and specialized SAT procedures exist

$\Rightarrow$ use SAT solvers for deciding more expressive logics and formalisms ...

$\Rightarrow$ ...reusing most of the work and knowledge available in SAT

SAT: The problem

A *literal* l is a proposition p or its negation $\neg p$. ($\neg\neg p \equiv p$)

Given the literals $l_1, \ldots, l_k$, a *clause* is $l_1 \vee \cdots \vee l_k$.

Given the clauses $c_1, \ldots, c_m$, a CNF (Conjunctive Normal Form) formula is $c_1 \wedge \cdots \wedge c_m$.

An *assignment*, or *valuation* v, is a partial function from the propositions to $\{\mathsf{TRUE}, \mathsf{FALSE}\}$.

We can extend the definition of v in the natural way to assign truth values to literals, clauses and formulas.

Given a CNF formula φ, we define the *propositional satisfiability problem (SAT)*:

Does there exist an assignment v to the propositions in φ such that φ is true?

SAT: Examples

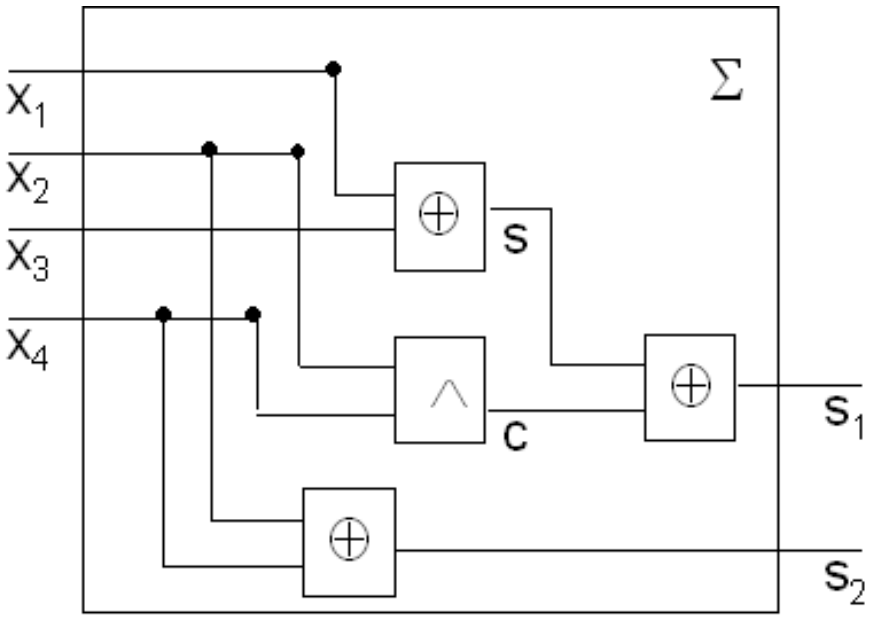
1. $\varphi := \{p, p \vee \neg q, \neg r\}$ has the satisfying (total) assignments
   - $\{p := \text{TRUE},\ q := \text{TRUE},\ r := \text{FALSE}\}$
   - $\{p := \text{TRUE},\ q := \text{FALSE},\ r := \text{FALSE}\}$

2. $\varphi := \{\neg p, p \vee \neg q, r \vee \neg p, q\}$ has no satisfying assignments because the clause $\{p \vee \neg q\}$ can not be satisfied.

The complexity of SAT

- Prototypical NP-complete problem
- n variables imply 2^n possible interpretations (total assignments) from which a satisfying one has to be guessed
- A SAT solver is a program that tries to find satisfying interpretations automatically

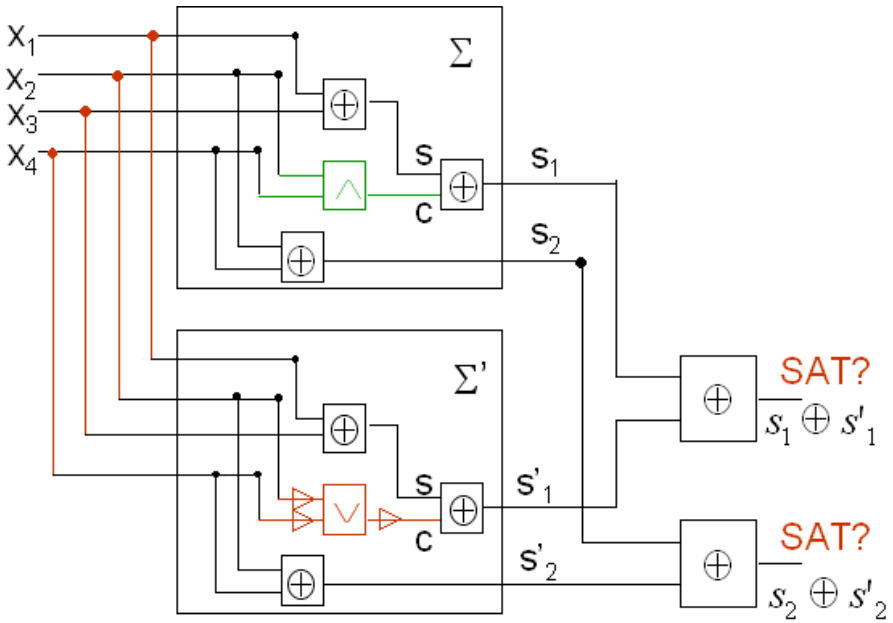
Formulas and circuits



$$s_1 = (\overbrace{(x_1 \oplus x_3)}^{s} \oplus \overbrace{(x_2 \wedge x_4)}^{c})$$
$$s_2 = (x_2 \oplus x_4)$$

input signal		variable
gate		operator
inverter	$\Leftrightarrow$	not
signal		subformula
output		formula value

Example I: Equivalence verification



Setting

- Σ is the abstract model
- Σ' is the implementation

Miter-based FCEV

SAT of $(s_1 \oplus s'_1) \vee (s_2 \oplus s'_2)$

Example II: AI Planning

Planning problem

Is a triple $\langle I, tr, G \rangle$ where (given the sets of fluents $\mathcal{F}$ and actions $\mathcal{A}$)

- I is a SAT formula over $\mathcal{F}$ and represents the set of *initial states*;
- tr is a SAT formula over $\mathcal{F} \cup \mathcal{A} \cup \mathcal{F}'$ where $\mathcal{F}' = \{f' : f \in \mathcal{F}\}$ is a copy of the fluent signature and represents the *transition relation*
- G is a SAT formula over $\mathcal{F}$ and represents the set of *goal states*.

Plan (Kautz and Selman. ECAI 1992)

The *planning problem* Π *with makespan* n is the SAT formula Π_n

$$I_0 \wedge \wedge_{i=1}^{n} tr_i \wedge G_n \qquad (n \geq 0) \tag{1}$$

- tr_i is the formula obtained from tr by substituting each symbol $p \in \mathcal{F} \cup \mathcal{A}$ with p_{i-1} and each $f \in \mathcal{F}'$ with f_i
- I_0 and G_n are obvious

A *plan for* Π_n is an interpretation satisfying (1).

SAT: Solving methods

- Resolution algorithm
- Local search algorithms
- (Ordered) Binary Decision Diagrams (OBDDs)
- Stalmark's method
- Davis-Logemann-Loveland (DLL) algorithm

Agenda

- DLL
- DLL-based (SAT-based) decision procedures
- Application I: Answer Set Programming (ASP)
- Application II: Separation Logic (SL) (a.k.a. Difference Logic)
- Application III: SAT-related optimization problems (Max-SAT, Min-ONE)
- Conclusions

The DLL algorithm

function SAT(ϕ) **return** DLL($cnf(\phi), \emptyset$);

function DLL(φ, S)
  if $\varphi = \emptyset$ **then return** TRUE ;
  if $\emptyset \in \varphi$ **then return** FALSE ;
  if $\{l\} \in \varphi$ **then return** DLL($\varphi_l, S \cup \{l\}$);
  $A :=$ an atom occurring in φ;
  return DLL($\varphi_A, S \cup \{A\}$) **or**
    DLL($\varphi_{\neg A}, S \cup \{\neg A\}$).

φ_l

φ_l returns the formula obtained from φ by (i) deleting the clauses containing l, and (ii) deleting $\neg l$ from the others.

Theorem (Davis, Logemann and Loveland, JACM 1962)

SAT(ϕ) returns TRUE if ϕ is satisfiable, and FALSE otherwise.

SAT-based decision procedures

Given a formula t in the theory T that can be abstracted/compiled into SAT

function *BEYSAT*(t) **return** DLL(*cnf*(T2SAT(t)), $\emptyset$);
function DLL(φ, S)
  if $\varphi = \emptyset$ **then return** $\underline{\mathit{test}(S, t)}$;
  if $\emptyset \in \varphi$ **then return** FALSE ;
  if $\{l\} \in \varphi$ **then return** DLL(φ_l, $S \cup \{l\}$);
  $A :=$ an atom occurring in φ;
  return DLL(φ_A, $S \cup \{A\}$) **or**
    DLL($\varphi_{\neg A}$, $S \cup \{\neg A\}$).

$\mathit{test}(S, t)$ returns TRUE if S is a "solution" for the formula t, and FALSE, otherwise.

Expected property

BEYSAT returns TRUE if t is satisfiable in T, and FALSE otherwise.

From DLL to BEY-SAT: Discussion

1. Most state-of-the-art (SOTA) SAT solvers are a (non-recursive) implementation of DLL
2. If SOTA SAT solvers are based on "learning" in order to backjump irrelevant nodes while backtracking and avoid the exploration of useless parts of the search tree, it is important that *test*(S, t) does not return only FALSE, but also a "witness" of inconsistency (called *reason*)
3. BEYSAT(t) works in polynomial space if both T2SAT and *test*(S, t) does
4. depending on T, BEYSAT(t) can be (easily) modified in order to compute all the solutions of t

Application I: Answer Set Programming (ASP)

Answer Set (stable model) Programming is a programming paradigm proposed by Marek, Truszczynski and Niemela in 1999.

It is a form of declarative programming. It is based on logic rules and on the answer set semantic of Prolog proposed by Gelfond and Lifschitz in 1988.

In answer set programming we obtain the answers by declaring the properties of the answers, by the mean of logic rules.

ASP has been used in several fields like planning, commonsense reasoning, (bounded) model checking, security protocols.

ASP: Logic rules for 3-colorability problem

Example

```
% Simple graph containing 3 nodes and 3 edges: edges
% between nodes 1 and 2, 2 and 3, 3 and 1, respectively.
node(1). node(2). node(3).
edge(1,2). edge(2,3). edge(3,1).
% Declaration of the three colors.
col(red). col(green). col(blue).
% A node has some color.
color(X,red) v color(X,green) v color(X,blue) :- node(X).
% Neighboring nodes should not have the same color.
:- edge(X,Y), color(X,C), color(Y,C), col(C).
```

Basic preliminaries

A *(logic) program* Π is a finite set of rules of the form

$$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n \tag{2}$$

Let P be the set of atoms in Π, $A_0 \in P \cup \{\bot\}$, $\{A_1, \ldots, A_n\} \subseteq P$. A_0 is the *head*. *not* is the negation as failure operator.

Comp(Π) consists of formulas of the type

$$A_0 \equiv \bigvee (A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n)$$

for each symbol in $P \cup \{\bot\}$. In the equation, the disjunction extends over all rules (2) in Π with head A_0.

What is an answer set?

Consider first Π in which $m = n$. Let X be a set of atoms.

We say that X is *closed* under Π if for every rule in Π, $A_0 \in X$ whenever $\{A_1, \ldots, A_m\} \subseteq X$.

We say that X is an answer set for Π if X is the smallest set closed under Π.

Consider now the general case $n > m$. The reduct Π^X of Π related to X is the set of rules

$$A_0 \leftarrow A_1, \ldots, A_m$$

such that $X \cap \{A_{m+1}, \ldots, A_n\} = \emptyset$.

We say that X is an answer set for Π if X is an answer set for Π^X.
problem.

ASP: Examples

1. Be Π_1: $p \leftarrow not\ q.$
   $q \leftarrow not\ r.$

The only AS is $\{q\}$

2. Be Π_2: $p \leftarrow not\ q$
   $q \leftarrow not\ p$

The ASs are $\{p\}$ and $\{q\}$

3. Be Π_3: $p \leftarrow not\ p$

Π_3 does not have AS.

Cmodels2 decision procedure

function CMODELS2(Π) **return** DLL($cnf(\underline{Comp(\Pi)}), \emptyset$);

function DLL(φ, S)
  if $\varphi = \emptyset$ **then return** $\underline{test(S, \Pi)}$;
  if $\emptyset \in \varphi$ **then return** FALSE ;
  if $\{l\} \in \varphi$ **then return** DLL($\varphi_l, S \cup \{l\}$);
  $A :=$ an atom occurring in φ;
  return DLL($\varphi_A, S \cup \{A\}$) **or**
    DLL($\varphi_{\neg A}, S \cup \{\neg A\}$).

test

$test(S, \Pi)$ returns TRUE if $S \cap P$ is an answer set of Π, and FALSE, otherwise.

Theorem (Giunchiglia, Maratea and Lierler; JAR 2006)

CMODELS2(Π) returns TRUE if Π has an answer set, and FALSE otherwise.

From DLL to Cmodels2: Discussion

1. CMODELS2(Π) can be easily modified in order to compute all the answer sets of a program Π
2. *test*(S, Π) can fail because of "loops" in Π. The reason is extracted from the "loop formulas"

Example

Π: $p \leftarrow p$, *Comp*(Π) is $p \equiv p$

3. CMODELS2 works in polynomial space

Experimental results: Blocks world

		Standard programs			Extended programs	
#b	#s	SMODELS	ASSAT	CMODELS2	SMODELS	CMODELS2
8	i-1	12.32	0.80	1.19	0.81	0.47
11	i-1	71.78	2.97	4.19	2.97	1.01
8	i	40.87	0.89	2.18	1.56	1.40
11	i	71.42	3.17	4.52	3.41	1.16
8	i+1	23.35	0.96	0.97	4.99	0.31
11	i+1	107.48	3.54	3.33	5.21	0.75

Table: Blocks world: “#b” is the number of blocks. “#i” the smallest number of steps for which a solution is found.

Experimental results: H.C. complete graphs

	Standard programs				Extended programs	
	SMODELS	ASSAT	DLV	CMODELS2	SMODELS	CMODELS2
np30c	11.70	1.14	22.08	0.69	0.36	0.36
np40c	62.89	41.81	97.96	1.63	2.48	0.87
np50c	219.56	14.51	314.46	3.37	8.39	1.79
np60c	594.46	48.80	770.07	5.81	20.47	3.41
np70c	1323.61	291.60	1679.12	8.22	39.41	5.87
np80c	2354.28	32.51	3407.35	14.20	75.36	9.18
np90c	TIME	779.06	TIME	22.23	122.53	14.19
np100c	TIME	–	TIME	28.63	185.52	20.76
np120c	TIME	–	TIME	53.33	418.15	41.84

Table: Complete graphs. npXc corresponds to a graph with “X” nodes.

Experimental results: Formal Verification problems

	SMODELS	ASSAT	DLV	CMODELS2
mutex4	33.92	0.62	840.60	0.68
phi4	0.24	2.98	1.44	TIME
mutex2	0.09	1.78		0.12
mutex3	229.57	MEM		24.16
phi3	2.87	236.91		3.91

Table: Checking requirements in a deterministic automaton.

Application II: Separation Logic (SL) (a.k.a. Difference Logic)

Decision procedures able to decide quantifier-free first-order theories are becoming increasingly important in Artificial Intelligence and Formal Verification.

Several properties of hardware, timed automata, and software can be modeled in quantifier-free first-order theories as well as planning and scheduling problems.

Separation Logic (SL) (a.k.a. Difference Logic) is one of such decidable quantifier-free first-order theories that allows boolean combination of difference constraints ($x - y \leq c$).

Two divisions of the SMT (Satisfiability Modulo Theories) Competitions are on Difference Logic.

Why Separation Logic?

SL is a good compromise between efficiency and expressivity.

It combines propositional atoms with a restricted form of linear arithmetic via the standard boolean connectives.

Many available benchmarks are in SL and a lot of properties of systems and planning/scheduling constraints can be encoded in this logic.

Example

1. "Activity A1 lasts for 10 units of time at most": $e_1 - s_1 \leq 10$
2. "Activity A1 should start before activity A2 finishes": $s_1 \leq e_2$
3. "Activity A1 should start before activity A2 finishes, otherwise A3 should start when A2 finishes": $s_1 \leq e_2 \vee s_3 = e_2$

SL: Definitions

Fix a domain of interpretation D for the arithmetic variables (the set of real or the set of integer numbers).

An SL-atom is either a propositional variable or a difference constraint $x - y \leq c$ ($<, >, \geq, =, \neq$ can be (easily) recast in $\leq$), where x and y range on D and c is a numeric constant.

An SL-literal is an SL-atom or its negation. An SL-clause is a finite disjunction of SL-literals. An SL-formula is a finite conjunction of SL-clauses.

Deciding the satisfiability of an SL-formula ψ means answer the question: "Is there an SL-assignment to propositional atoms and arithmetic variables, such that the SL-formula ψ is true?").

TSAT++ decision procedure

Given a formula ψ in SL,

function TSAT++(ψ) **return** DLL(*cnf*(*Abstract*(ψ)), $\emptyset$);
function DLL(φ, S)
  if $\varphi = \emptyset$ **then return** *test*(S, ψ);
  if $\emptyset \in \varphi$ **then return** FALSE ;
  if $\{l\} \in \varphi$ **then return** DLL(φ_l, $S \cup \{l\}$);
  $A :=$ an atom occurring in φ;
  return DLL(φ_A, $S \cup \{A\}$) **or**
    DLL($\varphi_{\neg A}$), $S \cup \{\neg A\}$).

test(S, ψ) returns TRUE if the set of constraints in S is consistent, FALSE otherwise.

Theorem (Armando, Castellini, Giunchiglia and Maratea; JAR 2005)

TSAT++(ψ) returns TRUE if ψ has a solution, and *False* otherwise.

From DLL to TSAT++: Discussion

- *test*(S, ψ) can fail because of sets of inconsistent difference constraints in ψ. The reason is extracted using the Bellman-Ford algorithm considering the difference constraints involved in a negative cycle.

- TSAT++ works in polynomial space

Disjunctive Temporal Problem (DTPs)

These are well-known random problems from the AI community. DTPs are randomly generated by fixing the number k of expressions $x - y \leq c$ per SL-clause, the number n of arithmetic variables, a positive integer L such that all the constants are taken in $[-L, L]$. Then:

1. the number of clauses m is increased in order to range from satisfiable to unsatisfiable instances from 2*n to 14*n step n,
2. for each tuple of values of the parameters, 100 instances are generated and then given to the solvers, and
3. the median of the CPU time is plotted against the m/n ratio.

TSAT++'s performance (1): DTPs

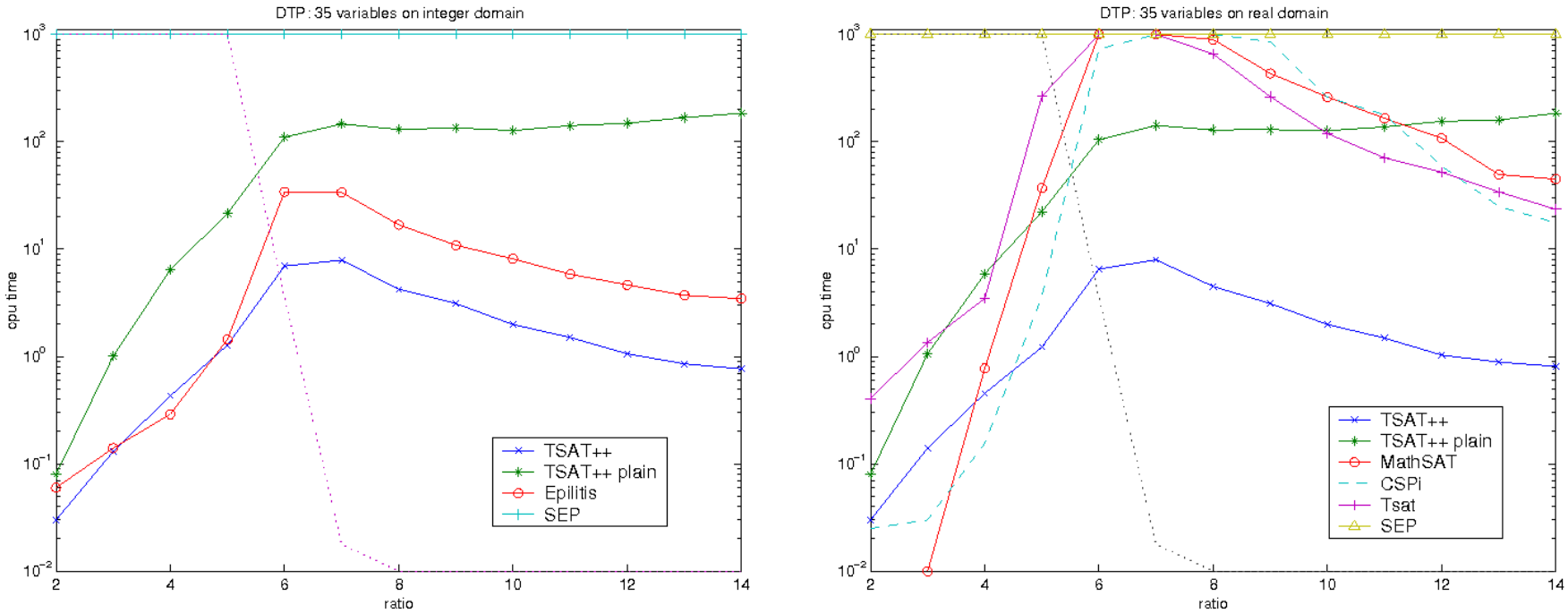


Figure: Evaluation on the DTP on 35 variables. Integer domain (left) and real domain (right). Setting: $k = 2$, $L = 100$.

TSAT++'s performance (2): hand-made problems

K	T	unique	TSAT++	TSAT++p	MathSAT	SEP
100	5	NO	0.01	0.11	0.61	1.18
100	5	YES	0.04	7.57	TIME	0.17
250	5	NO	0.08	0.76	5.40	52.20
250	5	YES	0.21	194.99	TIME	0.77
500	5	NO	0.29	4.46	21.22	742.99
500	5	YES	1.05	TIME	TIME	4.85
1000	5	NO	1.07	22.3	–	TIME
1000	5	YES	6.45	TIME	–	22.53
2000	5	NO	3.76	94.23	–	–
2000	5	YES	29.90	TIME	–	–

TSAT++'s performance (3): real-world problems

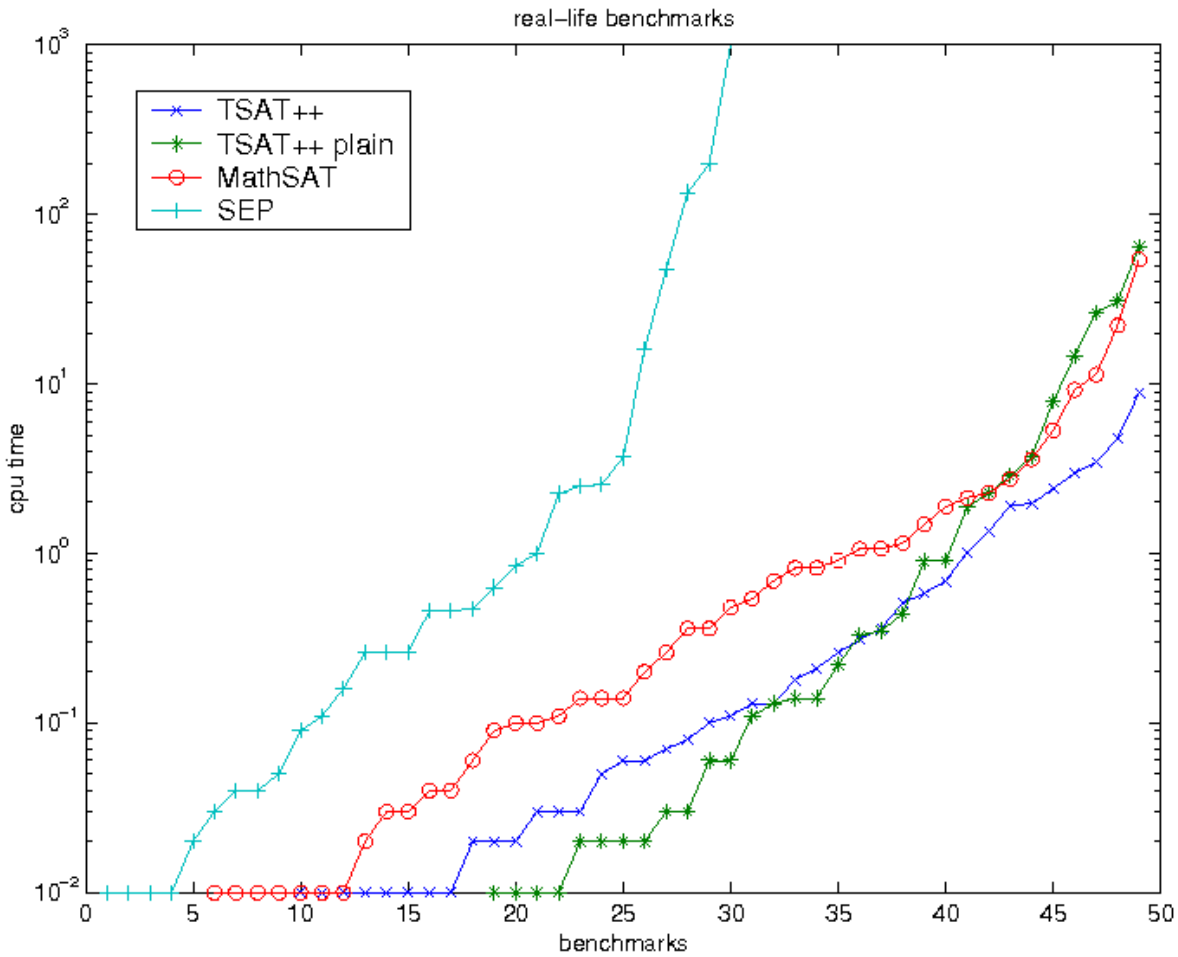


Figure: Real-world problems.

Application III: SAT-related optimization problems

There is some interesting research related to SAT, namely

1. Max-SAT: Given an unsatisfiable SAT instance, how many clauses can be satisfied at most (at the same time)?
2. Min-One: Given a satisfiable SAT instance, find the satisfying assignment with the maximum (minimum) number of variables assigned to TRUE

- These problems have applications in planning, model checking, correcting the minimum amount of inconsistent knowledge, and more.
- We propose an approach based on *preferences*.

Application III: Preferences

Preference

A (qualitative) preference is pair $G, \prec$ where

- G is a set of literals (the *set of preferences*) and intuitively represents the set of literals that we would like to have satisfied
- $\prec$ is a partial order on G and intuitively models the relative importance of our preferences.

Preference relation

Given two (total) models S and S', S *is preferred to* S' *(*$S \prec S'$*)* iff

1. there exists a literal $l \in S$ with $l \in S$ and $\bar{l} \in S'$; and
2. for each literal $l' \in S \cap (S' \setminus S)$, there exists a literal $l \in S \cap (S \setminus S')$ such that $l \prec l'$.

Optimal models

A model S of a formula φ is *optimal* if it is a minimal element of the partially ordered set of models of φ.

Application III: Example

Example

$\varphi = (\neg Fish \vee \neg Meat) \wedge (\neg RedWine \vee \neg WhiteWine)$
$\langle G, \prec \rangle = \{Fish, RedWine, WhiteWine\}, \{WhiteWine \prec RedWine\}$

The optimal model is $\{Fish, WhiteWine, \neg RedWine, \neg Meat\}$.

Preference formula

The *preference formula for S wrt* $G, \prec$ is

$$(\vee_{l:l \in G, l \notin S} l) \wedge (\wedge_{l':l' \in G, l' \in S} (\vee_{l:l \in G, l \notin S, l \prec l'} l \vee l')) \quad (3)$$

S' which satisfies (3) are such that $S' \prec S$.

Example

Considering the preference above

- if $S_1 = \{Meat, RedWine, \neg Fish, \neg WhiteWine\}$, then (3) is

$$\psi_1 : (Fish \vee WhiteWine) \wedge (WhiteWine \vee RedWine)$$

The SATPREF decision procedure

function PREF-DLL$(\varphi, G, \prec)$;$\psi := \top$; $S_{opt} := \emptyset$; **return** DLL$(\varphi \cup \psi, \emptyset)$;
function DLL$(\varphi \cup \psi, S)$
  if $\bot \in (\varphi \cup \psi)_S$ **then return** FALSE ;
  if *S is total Sopt* := S; ψ := *Reason*$(S, G, \prec)$; **return** FALSE ;
  if $\{l\} \in (\varphi \cup \psi)_S$ **then return** PREF-DLL$(\varphi \cup \psi)_S, S \cup \{l\})$;
  return PREF-DLL$(\varphi \cup \psi, S \cup \{A\})$ **or**
    PREF-DLL$(\varphi \cup \psi, S \cup \{\neg A\})$.

Reason$(S, G, \prec)$ returns the set of clauses corresponding to the preference formula for S wrt $G, \prec$;

Theorem (Di Rosa, Giunchiglia and Maratea; 2008)

PREF-DLL$(\varphi, G, \prec)$ terminates, and then *Sopt* is empty if φ is unsatisfiable, and an optimal model of φ wrt $G, \prec$ otherwise.

Issues in experimental analysis

Representation of the problems

Optimization problems are solved by setting preferences. Given P the set of atoms in φ, $\neg P = \{\neg p : p \in P\}$

- for MIN-ONE: $\langle G, \prec \rangle := \langle \neg P, \emptyset \rangle$
- for MAX-SAT: the problem is reduced to MIN-ONE bu using clause selectors

Quantitative preferences

A quantitative preference is a pair $\langle G, c \rangle$ where

- G is the set of preferences;
- c is a function mapping each literal into a positive integer number (intuitively the weight of the preference)

In our framework, quantitative preferences are reduced to qualitative.

Analysis on MAX-SAT problems

	domain	#I	OPTSAT	SATPREF	OPBDP	PBS4	MSAT+	BSOLO	MMSAT	OPTSAT	SATPREF
1	MINONE	26	0.69(26)	0.2(26)	85.37(7)	17.56(19)	7.33(24)	115.73(22)	87.21(24)	93.24(24)	23.99(25)
2	Partial MINONE	21	77.99(19)	2.7(21)	–	223.14(15)	43.32(18)	433.21(16)	391.21(12)	74.28(21)	69.89(21)
3	MAXSAT	35	26.68(34)	11.25(35)	20.89(3)	98.55(10)	130.37(31)	192.56(23)	229.73(21)	218.86(31)	175.12(31)
4	MAXCUT-1	5	0.01(5)	0.01(5)	0.99(1)	66.67(1)	0.86(1)	76.57(1)	1.09(3)	7.56(1)	7.52(1)
5	MAXCUT-2	62	0.01(62)	0.01(62)	230.33(5)	0.01(2)	247.54(7)	0.01(2)	194.52(52)	66.86(4)	21.61(3)
6	PSEUDO-1	7	0.02(7)	0.01(7)	2.2(4)	147.58(4)	0.25(5)	30.18(4)	4.75(5)	22.8(5)	36.66(5)
7	PSEUDO-2	17	0.03(17)	0.01(17)	–	85.88(1)	490.36(5)	–	81.93(2)	90.36(3)	338.26(3)
8	PSEUDO-3	148	4.81(130)	0.19(131)	16.65(85)	18.08(90)	11.52 (104)	22.23 (94)	62.08 (107)	31.8(103)	60.59(109)
9	PSEUDO-4	15	11.69(15)	3.12(15)	81.83(5)	102.75(9)	43.74(15)	373.73(8)	109.49(14)	41.49(15)	36.1(15)
10	MAXONE	60	0.96(60)	0.13(60)	296.26(35)	11.48(60)	2.02(58)	40.96(60)	22.5(60)	293(56)	7.87(58)
11	MAXCLIQUE	62	0.01(62)	0.06(62)	70.37(16)	23.79(13)	154.39(22)	248.26(14)	61.97(36)	54.14(19)	178.04(23)

Table: Results for solving satisfiability problems with qualitative (columns 4-5) and quantitative (columns 6-14) preferences. Problems are: MIN-ONE (row 1), partial MIN-ONE (row 2), MAX-SAT (rows 3-5), and partial MAX-SAT (rows 6-11).

Conclusions

- the SAT-based approach is very competitive and often superior w.r.t. rival approaches in a number of application areas
- it can leverage on the work done on SAT in the last years, and our algorithms can (easily) take advantages on future enhancement in the field (SAT competitions every year!)
- here we have focused on NP-complete problem, but the approach
  can be/has been applied to “harder” problems such as disjunctive logic programming and conformant planning

References

- E. Di Rosa, E. Giunchiglia, M. Maratea - Solving Optimization Problems with DLL: A new Approach. Submitted to ECAI 2008.
- E. Giunchiglia, Yu. Lierler, M. Maratea - Answer Set Programming based on Propositional Satisfiability. Journal of Automated Reasoning (JAR), Vol. 36(4), pgg. 345-377 (2006).
- A. Armando, C. Castellini, E. Giunchiglia and M. Maratea - The SAT-based Approach to Separation Logic. Journal of Automated Reasoning (JAR), Vol. 35(1-3), pgg. 237-263 (2005).

Resolution method

- based on resolution rule: $l \vee C_1$ and $\neg l \vee C_2$ resolve into $C_1 \vee C_2$
- is the "root" of DLL ancestor (DP)
- resolution based theorem provers are usually targeted for first-order logic
- DP variable elimination rule has a number of disadvantages w.r.t. DLL splitting

Local search algorithms

- randomly select an assignment for φ
- then try to minimize the unsatisfied clauses
- can not guarantee completeness

OBDDs

- introduced by (Bryant 1992)
- boolean functions are represented via directed acyclic graphs
- in the worst case the graph is exponentially larger (in the number of variables)
- some operations on the graph and between graphs are very convenient
- (ordered): introduces a total order on the variables
- highly dependent on the ordering of the variables in the graph

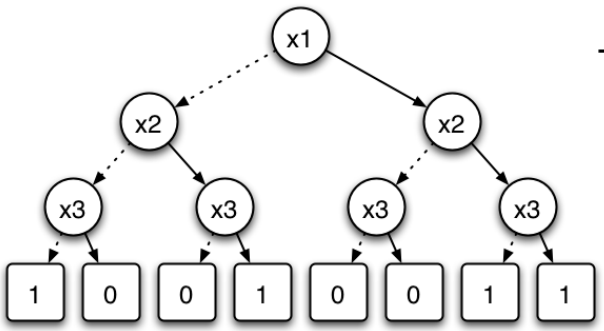
Stalmarck's method

- patented proof method developed by Gunnar Stalmarck (1989)
- it is based on a system for natural deduction
- bread-first backtracking algorithm
- commercial tool Prover and SAT solver Heerhugo are based on this method
- solver Heerhugo can actually deal with more than propositional logic

Diamonds problems

Given a parameter K (number of diamonds), these problems are characterized by an exponentially large (2^K) number of boolean models T, some of which correspond to satisfying SL-assignments; hard instances with a unique satisfying SL-assignment can be generated.
A second parameter, T (related to the number of edge in each diamond), is used to make T larger, further increasing the difficulty.
Variables range over the reals.

BDD - Example



x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

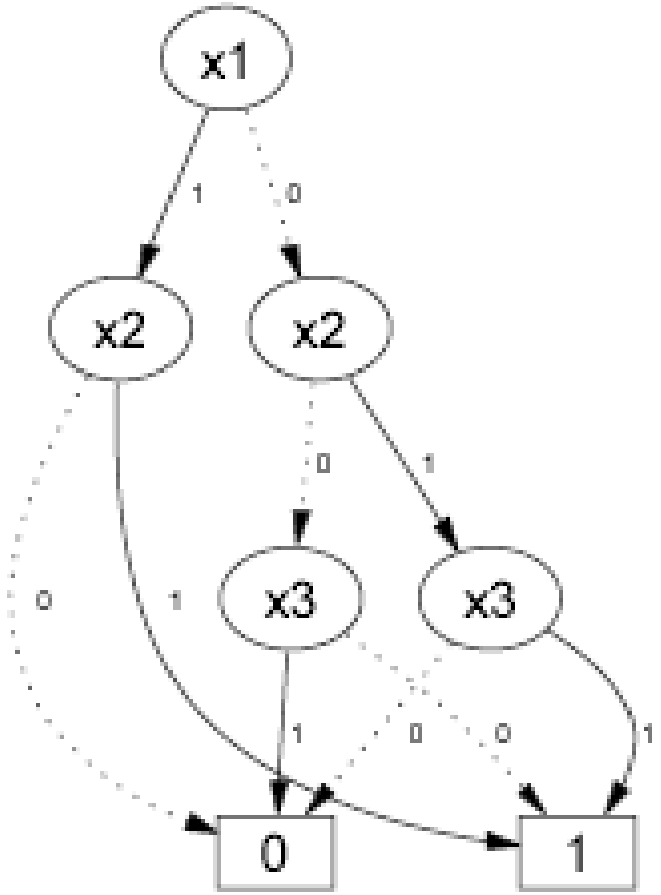


Figure: BDD for the boolean function $f(x1, x2, x3) = \neg x1 \times \neg x2x3 + x1 \times x2 + x2 \times x3$. Binary decision tree (left) and binary decision diagram (right).